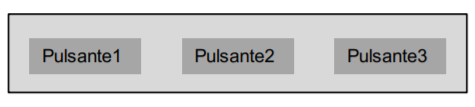
**Prova scritta del 06/2017**

**1 – Si consideri il LinearLayout indicato in figura. Si costruisca tale layout utilizzando un file statico che specifica i primi 2 pulsanti, ed inserendo dinamicamente il terzo pulsante (dettagliare il file XML e lo snippet di codice che serve ad aggiungere il terzo pulsante).**

****

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 android:id="@+id/linear">  
  
 <Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Pulsante1" />  
  
 <Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Pulsante2" />  
  
 </LinearLayout>  
  
</RelativeLayout>

public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 LinearLayout linearLayout = (LinearLayout) findViewById(R.id.*linear*);  
 Button b = new Button(this);  
 b.setText("Pulsante3");  
 linearLayout.addView(b);  
 }  
}

**2 – Si consideri il seguente frammento di codice che definisce un OnItemClickListener per un ListView.**

**listView.setOnItemClickListener(new OnItemClickListener() {  
 @Override  
 public void onItemClick(AdapterView <?> parent, View view, int  
 position, long id) {  
 String str = listView.getItemAtPosition(position).toString();  
 // Fai qualcosa…  
 }  
});**

**Si spieghi il ruolo del parametro position.**

La classe OnItemClickListener viene utilizzata allo scopo di gestire il click; il suo metodo onItemClick conterrà il vero codice da attivare ad ogni selezione di elemento. I suoi parametri in input conterranno tutte le informazioni utili tra cui: l’elemento su cui si è cliccato (il secondo, di classe View); la posizione che ricopre nella struttura (il terzo parametro, di tipo int, ovvero position). All’interno del metodo onItemClick si vede come l’informazione della posizione è stata sfruttata per recuperare dalla struttura dati l’oggetto.

**3- Si consideri la seguente situazione: un’app viene lanciata e l’utente interagisce con l’app; ad un certo punto l’utente ruota il dispositivo e continua ad interagire con l’app; infine, l’utente chiude l’app tramite un apposito pulsante. In quali stati è passa l’app e quali metodo sono stati chiamati dal momento in cui l’app viene lanciata al momento in cui termina?**

onCreate() -> onStart() -> onResume() -> onPause() -> onStop() -> onDestroy() -> onCreate() -> onStart() -> onResume() -> onPause() -> onStop() -> onDestroy().

**4 – Stati usando il tuo smartphone per fare dei conti con un’app “calcolatrice”, come mostrato nella figura a sinistra. Involontariamente ruoti il dispositivo e ti ritrovi con la situazione descritta nella figura a destra. Il valore “783” è diventato “0” e la disposizione dei tasti è cambiata. Cosa è successo? Cosa ha fatto bene e cosa ha sbagliato il programmatore dell’app? Cosa avrebbe dovuto fare per ovviare all’errore?**

Come possiamo notare, la disposizione dei tasti durante la visualizzazione landscape è diversa rispetto alla loro disposizione durante alla visualizzazione portait. Questo avviene perché il programmatore ha creato due file xml, uno per la visualizzazione portait e uno per la visualizzazione landscape.

D’altro canto, però, il programmatore non ha gestito la perdita di stato durante la rotazione dello schermo. Come possiamo notare, il numero 783 durante la rotazione viene perso, questo perché, quando lo schermo viene ruotato, l’acitivity viene distrutta e ricreata, questo appunto viene fatto per poterla adattare meglio allo schermo. Così facendo però si perde anche lo stato, e, per ovviare al problema, ci basterà sovrascrivere il metodo onSaveInstanceState(Bundle b) che verrà chiamato prima della distruzione dell’activity. In questo metodo ci basterà salvare lo stato dell’applicazione nell’oggetto bundle (l’oggetto bundle è una mappa, ovvero contiene coppie chiave/valore che vengono salvate con il metodo put e prelevate con il metodo get). Lo stato verrà recuperato nel metodo onCreate, infatti, basterà fare un controllo sull’oggetto bundle. Se quest’ultimo non è vuoto, ripristiniamo il vecchio stato dell’applicazione.

**5 – Si consideri il seguente snippet di codice dell’activity ActivityA:**

**Intent i = new Intent(getBaseContext(), ActivityB.class);  
i.putExtra(“NOME”, “Roberto”);  
i.putExtra(“VALORE”, 23);  
i.putExtra(“STATO”, true);  
startActivity(i);**

**Si scriva lo snippet di codice per l’activity B per recuperare i valori inseriti nell’intent.**

Intent i = getIntent();  
String nome = i.getStringExtra(“NOME”);  
Int valore = i.getIntExtra(“VALORE”);  
boolean stato = i.getBooleanExtra(“STATO”);

**6 – Per quali situazioni occorre utilizzare la classe AsyncTask? Si faccia un esempio.**

Per sviluppare applicazioni veloci, è necessario delegare le operazioni “lente” a thread secondari, così da poter velocizzare l’applicazione. Inoltre, operazioni che riguardano l’accesso alla rete, per una buona programmazione Android, devono essere delegate a thread secondari.  
I thread secondari però, non possono in nessun modo, modificare l’interfaccia grafica. Per questo motivo, ogni qualvolta un thread secondario vuole interagire con l’interfaccia grafica, deve farlo per via del thread main. Ciò implica che i thread secondari devono in qualche modo comunicare con il thread main. Per questo motivo esiste la classe AsyncTask. Grazie a questa classe, il thread secondario può svolgere tranquillamente le sue operazioni, ed ogni qualvolta lo desidera, grazie al metodo publishProgress, comunica il suo stato al thread main. A questo punto il thread main può interagire con l’interfaccia grafica in base allo stato di avanzamento del thread background.

Facciamo un esempio:  
Immaginiamo che al click di un pulsante viene caricata un’immagine e iniettata all’interno di un ImageView. Poiché questa operazione potrebbe richiedere del tempo, decidiamo di delegare ad un thread secondario questa mansione. Grazie alla classe AsyncTask, possiamo far si che, durante il caricamento dell’immagine, il thread background può comunicare lo stato di caricamento al thread main. Il thread main in questo caso, può mostrare sull’interfaccia grafica una progress bar che segna lo stato di avanzamento del caricamento dell’immagine.

**7 – Cosa è l’albero delle view? Quali sono e come funzionano le due fasi necessarie per la visualizzazione del layout?**

L’albero delle view rappresenta la gerarchia delle view. Le fasi necessarie per la visualizzazione del layout sono 3:  
- Fase di Misurazione (onMeasure()): nella fase di misurazione c’è un processo di negoziazione tra parent e figli. I figli definiscono una measuredSize ovvero quanto grande la view dovrebbe essere, che poi si trasformerà nella size reale: quando grande la view sarà in realtà.  
Ogni view può esprimere la propria preferenza usando la classe ViewGroup.LayoutParams. La misurazione avviene nel metodo onMeasure.  
- Fase di Layout (onLayout()): la View Parent decide la grandezza e la posizione dei figli in accordo alle misure fatte nella fase precedente.  
- Fase di Disegno (onDraw()): la view viene disegnata col metodo onDraw(), Quando c’è un cambiamento si utilizza invalidate() chiamando onDraw() su tutta la view e requestLayout() che ripete l’intero processo su tutto l’albero.

**8 – Come funzionano le animazioni? Si faccia un esempio.**

Android permette di definire delle animazioni da applicare alle immagini descrivendole tramite file XML o programmaticamente. Le animazioni disponibili sono: rotation, translate, scale, alpha…

Esempio di animazione:

<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android">  
  
 <translate  
 android:startOffset="0"  
 android:fromXDelta="0"  
 android:toXDelta="500"  
 android:duration="1000" />  
  
 <translate  
 android:startOffset="1000"  
 android:duration="2000"  
 android:fromYDelta="0"  
 android:toYDelta="300" />  
  
</set>

Una volta descritta l’animazione, per poterla utilizzare, basterà utilizzare il metodo setAnimation del riferimento che vogliamo animare:

b.setAnimation(AnimationUtils.*loadAnimation*(getApplicationContext(), R.anim.*animazione*));

**9 – Si descriva l’utilizzo di un sensore spiegando cosa si deve fare per utilizzarlo. Arricchire la spiegazione con porzioni di codice.**

Molti smartphone, tablet hanno sensori di movimento, di ambiente, di posizione etc… I sensori forniscono dati “grezzi” e l’accuratezza dei dati dipende dalla qualità.

Un SensorManager ci dice i sensori disponibili e le caratteristiche del singolo sensore. Inoltre, ci permette di leggere i dati del sensore e usare Listeners sui cambiamenti dei dati.

Per ottenere un riferimento al SensorManager bisogna usare il metodo  
getSystemService(SENSOR\_SERVICE);

Grazie al SensorManager possiamo ottenere riferimenti ai vari sensori. Ad esempio, per ottenere un riferimento all’accelerometro bisognerà fare:  
sensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER);

I listener che gestiscono i cambiamenti di stato del sensore devono essere registrati nel metodo onResume e rilasciati nel metodo onPause. Questo per evitare un consumo di batteria inutile.

Protected void onResume() {  
super.onResume();  
sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR\_DELAY\_UI);  
}

Protected void onPause() {  
super.onPause();  
sensorManager.unregisterListener(this);  
}

Per gesitere gli eventi, la classe deve implementare l’interfaccia SensorEventListener e quindi anche i suoi metodi.

Nel metodo onSensorChanged(SensorEvent event), possiamo gestire il cambiamento di stato del sensore tramite l’oggetto SensorEvent.

**10 – Si discuta del meccanismo dei frammenti. Si faccia un esempio per illustrane l’utilità.**

Un frammento è una porzione di Activity. Non si tratto solo di un gruppo di controlli o di una sezione del layout. Può essere definito più come una specie di sub-activity con un suo ruolo funzionale molto importante ed un suo ciclo di vita.

Un frammento non può vivere senza un’activity. Come detto il frammento ha il suo ciclo di vita fortemente collegato con quello dell’activity di appartenenza.

Un’activity può ospitare vari frammenti. I frammenti possono essere inseriti e rimossi durante l’esecuzione. Si possono creare UI con molti frammenti anche in funzione della grandezza dello schermo.

La filosofia di progettazione che risiede dietro i frammenti riguarda lo sviluppo di un’interfaccia utente dinamica, in particolare capace di adattarsi sia a schermi grandi che a schermi piccoli.

Un frammento può comunicare con l’activity creando dei metodi di callback.